CSci 4271W (010 Section) Project Instructions

Project 2 **Due: April 10th and April 15th, 2025**

1 Project: Badly Coded Bookmark Manager (BCBM)

As you know from project 1, Badly Coded, Inc., (BCI) is a software vendor responsible for keeping security students at UMN busily employed for many years. (Ask your friends! They're known for such incredible products as the "Badly Coded Versioning System (bcvs)" the "Badly Coded Compression System (bczip)", the "Badly Coded Print Daemon (bcpd)", and the "Badly Coded Calendar Alert System (bccal)" to name but a few. . .) The latest product under development at BCI is the Badly Coded Bookmark Manager (bcbm, for short), and the team responsible for this project has approached your group for a security assessment of (the pre-alpha version of) this code.

bcbm is intended to be a cross-browser and cross-device bookmark manager system: it can be run once on a user's device, and accessed from any browser on the device. Using the browser interface, users can add bookmarks, delete them, and order them in a "top 10" list of favorite bookmarks. bcbm can also synchronize a user's bookmarks across multiple devices, by uploading and downloading encrypted copies of the user's bookmarks file to a back-end bcbm cloud server. This synchronization between devices is managed through an email registration process: users enter an email address and password in the client that is submitted to the cloud server when uploading and downloading the bookmarks file; to set up an account, users visit a page on the cloud server, enter their email address, and are sent a confirmation link that can be used to set or reset the password.

Functionally, bcbm itself consists of two codebases:

- The bcbm "client", `bcbmc` is actually a small HTTP application server program (built using the CrowCPP framework) that runs on a user's device. Users connect to the client from a web browser, and the client writes web pages that accept user inputs and contain the results of user actions, which are then displayed in the browser. The client also handles the client-side portion of synchronizing between multiple devices and communicates with. . .
- A cloud server, `bcbms` for facilitating synchronization between devices. The cloud server program has functionality to transfer encrypted bookmark files, verify user credentials, and register or update credentials using registration codes.

However, a number of other entities are potentially involved: `bcbmc` connects to third-party websites to download website icons and web page titles; the user's interactions are sent through one or more web browsers, and the browser (typically) displays pages from other web sites that could potentially try to issue commands to the client through included resources. As you progress through the stages of the project, you'll see more and more of the code for bcbm, but these other entities, and the communication with them, should also be considered as you think about the potential threats to the client and server.

2 Project 2: Client `bcbmc` and "cloud" `bcbms`

For the purpose of the project, we're considering the bcbm code in three stages. For project 2, we'll consider the full client code, along with the portion of the server code that handles synchronization, but no authentication or registration. (We'll add the server code that handles authentication and registration in project 3)

To install the full `bcbmc` client and the (partial) `bcbms` server on your VM (please don't install this on a real machine - it's really buggy code) follow these steps:

1. Clone the source repos using:

```
git clone https://github.umn.edu/badly-coded-alpha/bcbmc-sync.git
git clone https://github.umn.edu/badly-coded-alpha/bcbms.git
```

2. Build the client by changing directory into `bcbmc-sync` and running `./install.sh`; then build the server by changing directory back to `../bcbms` and running `./install.sh`.

This will build and install the full `bcbmc` executable, as well as the partial `bcbms` server. Interactions with `bcbmc` continue to work as in project 1, although you should be careful for project 2 to make sure you're running the new executable, rather than the project 1 executable. To help with this we've added a logging message to the output and updated the title of the `bcbmc` "main" page to include the string "pre-alpha2" to indicate the newer client is being used.

`bcbms` is installed as a "systemd service", which means the operating system monitors the process, restarting it if it crashes, and it runs as `root`. To start `bcbms` on your VM, type `sudo systemctl start bcbms`, and to stop it run `sudo systemctl stop bcbms`. You can view the output logs of `bcbms` with `sudo journalctl -u bcbms`. The install.sh script also configures your VM (but <u>only</u> your VM) to route the hostname `bcbm.badlycoded.net` back to itself, so only a `bcbmc` instance running on your VM will be able to connect to the server running on your VM.

The code for bcbmc is split across several files:

- The main code for the application server is in `bcbmc.cpp`, which uses the CrowCPP framework. The framework itself (the code in `crow_all.h`) should be excluded from your analysis, but you should at least skim the documentation for CrowCPP to help you read and understand the code in `bcbmc.cpp`. The `main` function is in `bcbmc.cpp` and uses Crow to "route" requests to the other modules in the system. It also calls functions to set the six-digit "authentication" code that (attempts to?) prevent(s) other pages/programs from calling the bookmark management functions of `bcbmc`.
- The file `acode.c` contains the code for choosing a random authentication code.
- The file `manage.c` contains most of the code for responding to user requests. For most interactions, a function is called from `bcbmc.cpp` with the inputs provided by the user, and the function writes an html file with results to be displayed. `bcmbc.cpp` then returns this file to the browser for display.
- The file `sync.c` now includes all of the code for synchronizing bookmarks across devices. This includes connecting to the cloud server to download the latest archive (consisting of an encrypted bookmark file, public encryption keys for each device connected to the user's account, and the icons for each bookmarked site) of the user's bookmark file; decrypting the encrypted bookmark file using a locally-stored decryption key for the device, applying any changes in the local transaction file (`bcbm.tx`) to the cloud version of the bookmark file; re-encrypting the patched bookmark file under a randomly-chosen symmetric key, then encrypting that symmetric key under each device public key, making a new archive, and uploading the new archive to the cloud server.

The code for `bcbms` is also split across several files:

- The main code for the cloud server is in `bcbms.cpp`. Like the client, this is a CrowCPP application that "routes" requests to various back-end functions.
- The `cloudsync.cpp` file handles the cloud side of synchronization, verifying uploaded archives and then moving them to a common directory; and returning archives for downloading on request. It uses the `RegDB` class to check that an upload/download is from a registered, authorized user, although for <u>this</u> iteration of the project the check always succeeds: note that this should not be considered a vulnerability, since its exclusion is deliberate for testing purposes.

- The RegDB class is defined in regdb.h. Eventually, this class will both handle checking of username/password requests, as well as using an email registration code system to allow users to set/re-set their passwords. For now, as a testing measure, it simply returns true on any name/password pair. (Note again: because this is not intended to be deployed code, it's not a security vulnerability.)

Some other notes about interacting with/testing bcbmc and bcbms for Project 2:

- A possible threat that the developers clearly considered is tampering/spoofing of the user's commands by other websites displayed in the browser. This is a possibility because any website can tell the browser to load certain "resources," commonly images, from another website. A simple example of this is the file:

  ```
  <html><img src="http://localhost:8888/666666/add/http%3A%2F%2Fevil.com%2F"></html>
  ```

  which would cause the browser to send an "add" command for http://evil.com/ to their local bcbmc client. (Thanks to the randomized authentication string, the command will almost certainly be rejected!) You can experiment with similar files as examples of this type of threat: local files can be loaded in the browser. For testing, you can also use the curl command line tool (on your VM) to send a request to bcbmc and observe the response:

  ```
  % curl "http://localhost:8888/666666/add/http%3A%2F%2Fevil.com%2F"
  ```

  (Since bcbmc itself uses curl you might benefit from spending some time reading its man pages)

- bcbmc also interacts with web pages when adding them as bookmarks. For testing purposes, you can place pages in your CSELabs user homepage and add/delete them as bookmarks as you see fit. A PoC exploit using a page from one of these directories should include the code for the page, and a curl command to call the corresponding action in the client.

- **NO SOCIAL ENGINEERING!** The purpose of this project is to find problems inherent in the code of bcbm; so attacks that involve asking the user to run bcbmc with odd arguments, or paste strange values into bcbmc are out of scope. (However, it is OK to assume that the user might bookmark an attacker-controlled page or load an attacker-controlled page in their browser.)

- While there are several Elevation of Privilege attacks planted in this code, and many more may be possible due to the fact that it is Badly Coded, don't forget to look for other possible STRIDE threats, such as potential spoofing, tampering, information disclosure and denial of service against the client server. Your final report, after all, should include at least two vulnerabilities that are not in the EoP category.

- The use of encryption on the bookmarks file indicates the developers were also concerned with leaking potentially sensitive information about users' bookmarks to the cloud server (for example, imagine if a user had a bookmark to alcoholicsanonymous.com or plannedparenthood.org). It would certainly be worth investigating whether the measures taken here are effective.

- As with bcbmc, you can experiment with how bcbms responds to requests using curl; the bcmbc-sync code seems to use curl for its interactions with the server, so you might start by looking at those examples to formulate ideas about attacks that focus on the server side.

- **Note about DoS:** Since bcbms is a network server, availability is clearly an important feature, and so denial of service is certainly a consideration. However, bcbms is installed and started with a script that detects when the server process crashes and restarts the server. So a crash bug that doesn't impact the ability of the server to restart normally and resume functioning is more of a bug than a

DoS vulnerability. Also keep in mind that there's nothing a programmer can do to prevent network-level DoS or DDoS: please exclude these from your threat model and vulnerability reports. (The U's OITSec will get quite irate with us if someone tries to DDoS the network for their course project, so don't do that.)

- Because only your VM will know to route requests to `bcbm.badlycoded.net` back to the VM, using port forwarding to interact with `bcbmc` in a browser not running on the VM will not work to load the pages hosted on the `bcbms` server. In this version of the code, there's only one user-visible page, and it doesn't yet do anything (`https://bcbm.badlycoded.net/reg/start`) but if you want to see this page, you'll have to load it from a browser running on your VM. Thinking as an attacker, you shouldn't need to concern yourself with the user interface, but in Project 3, we'll include some suggestions for accessing the UI through a local (non-VM hosted) browser.

If you have questions about getting `bcbmc` or `bcbms` to work, or about the wording of the assignment, or about finding someone to work with, use the `project` folder on Piazza so that others can benefit from the answers to your questions. If you have any other questions about what is allowed for the project or other details, please ask the instructor or a TA as early as possible.

3 Assignment and Timeline

As also explained on the Project overview page, your assignment is to review and produce a security assessment of the BCBM code so far. For Project 2, you should produce a report that includes (1) a system design section that answers the question "what are we building?", (2) a threat model using STRIDE that outlines the types of threats you evaluated the system against, and (3) a summary of findings that documents and explains at least 4 vulnerabilities, at least one of which should be in the code released for project 2. You will also produce "proof of concept" programs showing that your vulnerabilities can be exploited, and a `README.md` file explaining how to run your proof of concept programs and what to expect. Finally, your report should have a separate section entitled "Addendum: Revisions and Group Accountability" which details which sections were primarily drafted or revised by each group member along with how you have revised the portions of your report based on the feedback you received for Project 1 (if you're working on individually, this will just cover revisions). To demonstrate the separate efforts of each group member, you will need to individually submit drafts of a section you have been working on before each full group submission. Recall that each student in a group should individually be the primary author, over the course of the semester, of 2 vulnerability subsections, and one design section or revision, and one threat model section or revision, so you should rotate responsibiliies for drafts compared to project 1.

3.1 Grading

- As noted on the project information page,
  - 20 points are assigned for including the required elements. Check that you included: design section, DFD, all DFD elements, threat model section, listing of security goals in the threat model, STRIDE by each element in the DFD, summary of findings, 4 vulnerabilities, mitigation discussions for all vulnerabilities;
  - (For Project 2) 5 points are assigned for the per-student section draft;
  - (For Project 2) 5 points are assigned for the "revisions and group accountability" addendum; did you address the feedback you received from Project 1?

  - 30 points are assigned for high-quality writing, as described in a separate writing rubric web page.
  - 16 points are split between the four vulnerabilities in your project report: for each vulnerability, did you clearly describe an actual vulnerability, correctly explain how it could be mitigated, clearly distinguish it from the other vulnerabilities?

  - 8 points are available for the clarity and correctness of the `README.md` file in your proof of concept repo.
  - 16 points total are available for the readability, reproducibility, and correctness of the proof-of-concept programs in your repo: 2 each for the revised PoCs from project 1, and 6 each for the two new PoCs (at least one targeting project 2 code.)

- To re-emphasize an important point from the project information page: **Claiming a vulnerability without confirming your claims about it is academic and professional dishonesty and may result in a grade of 0 for the project**. (An example would be to claim that a stack buffer in the function `gen_auth_code` can be overflowed, resulting in a tampering attack, without giving any evidence that an input from an untrusted source, of the appropriate length, could actually reach the `gen_auth_code` function. If all you are sure of is that the code looks risky but you're not sure an attack is possible, you can get partial credit for saying that honestly.)

3.2 Timeline:

- **By 11:59pm, Thursday, April 10th**: Each member of a group will submit a PDF draft of one section of the report that they have been responsible for. This submission isn't for detailed grading or feedback, but to confirm that each member of the group has been working on part of the report.

- **By 11:59pm, Monday, April 14th**: If you are requesting an extension to Friday April 28th, for project 2 (thereby using your only extension request) you should send email to Prof. McCamant with title **"4271: Project 1 extension request"** and list your name and your group members, along with a copy of your current report draft.

- **By 11:59pm, Tuesday, April 15**: You should submit your PDF report to Gradescope. The report should also include a clearly labeled reference to a private **UMN** github repo (so, on `github.umn.edu`) that is accessible to `smccaman` and `foiso001`, where we can find your two (or more) proof of concept programs and `"README.md"` file.

Happy Hacking!